The exam-randomizechoices package

LaTeX package for creating random placed choices in multiple choice environments using the exam document class

Jesse op den Brouw
Department of Electrical Engineering
The Hague University of Applied Sciences
Delft, Netherlands
J.E.J.opdenBrouw@hhs.nl

July 31, 2021

This is the user's guide for version 0.2 [July 31, 2021] of the exam-randomizechoices package.

Contents

1 Introduction

This document describes the LaTeX `exam-randomizechoices` package. The package provides the user with four new multiple choice typesetting environments which place the content in a random order. It can (only) be used in combination with the `exam` document class. It can only randomize the placement of choices in multiple choice questions. The questions themselves can't be randomized with this package.

Furthermore, the package provides a simple answer key table typesetter and has a command for writing the answer keys to an external file.

1.1 License and warranty

This work may be distributed and/or modified under the conditions of the LaTeX Project Public License, either version 1.3 of this license or (at your option) any later version. The latest version of this license is in `http://www.latex-project.org/lppl.txt` and version 1.3 or later is part of all distributions of LaTeX version 2003/12/01 or later.

This work has the LPPL maintenance status "author-maintained".

This work consists of the files `exam-randomizechoices.sty`, `exam-randomizechoices.tex` and `exam-randomizechoices-doc.tex`

This software is provided 'as is', without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

1.2 Where to find

The development version of this package is available on GitHub. See

`https://github.com/jesseopdenbrouw/exam-randomizechoices`

1.3 Acknowledgements

The author wishes to thank the developers of the `exam` document class and the `mcexam`, `environ`, `etoolbox` and `pgffor` packages.

1.4 Warning

This package is experimental, so it could possibly break LaTeX compilation. Use this package with care. Please report any problems to the author.

Please use a recent version of the `exam` document class. This package is tested with version 2.603 which is available in most distributions. Testing with version 2.606β is planned.

1.5 A word about reading this document

This document uses terminology which is described below:

- If you encounter the term "the standard multiple choice environments", you should read this as "the `choices`, `oneparchoices`, `checkboxes` and `oneparcheckboxes` environments".
- If you encounter the term "the new multiple choice environments", you should read this as "the `randomizechoices`, `randomizeoneparchoices`, `randomizecheckboxes`

and `randomizeoneparcheckboxes` environments".

- If you encounter the term "`*choices`", you should read this as "`choices`, `oneparchoices`, `randomizechoices` and `randomizeoneparchoices`".
- If you encounter the term "`*checkboxes`", you should read this as "`checkboxes`, `oneparcheckboxes`, `randomizecheckboxes` and `randomizeoneparcheckboxes`".
- If you encounter the term "command" in the context of a backslash followed by a series of characters, you can read this as "macro". The official TEX-name is "control sequence".

2 Using the `exam` class

This section provides a limited introduction to the `exam` document class. As a novice user, please read on. If you are a experienced user, you may skip this section.

The `exam` document class is a powerful class to create exams with LATEX. Both open questions and multiple choice questions are supported. For multiple choice questions we can differentiate between enumerated lists or checkbox lists. The class documentation states[1]:

> The file `exam.cls` provides the `exam` document class, which attempts to make it easy for even a LATEX novice to prepare exams. Specifically, `exam.cls` sets the page layout so that there are one inch margins all around (no matter what size paper you're using) and provides commands that make it easy to format questions, create flexible headers and footers, change the margins, and create grading tables. In more detail:
>
> - The class will automatically format and number the questions, parts of questions, subparts of parts, and subsubparts of subparts.
> - You can include the point value of each question (or part, or subpart, or subsubpart), with your choice of having the point values printed at the beginning of the text of the question, opposite that in the left margin, opposite that in the right margin, or in the right margin opposite the end of the question.
> - The class will add up the total points for each question (and all of its parts, subparts, and subsubparts) and the total points on each page, and make those totals available in macros.
> - You can have the class print a grading table, indexed either by question number or by page number.
> - You specify the header in three parts: One part to be left justified, one part to be centered, and one part to be right justified, and one or all of these can be omitted.
> - The footer is also specified in three parts: Left justified, centered, and right justified.

[1] See https://ctan.org/tex-archive/macros/latex/contrib/exam.

- The header and footer for the first page can be different from the ones used on other pages.
- Both headers and footers can contain more than one line. To accommodate headers and footers with several lines, simple commands are provided to enlarge the part of the page devoted to the header and/or footer, and these commands can give one amount of space on the first page and a different amount of space on all other pages.
- Macros are defined to enable you to state the total number of pages in the exam and to change the header and/or footer that appears on the last page of the exam .
- Macros are defined so that the headers and footers can vary depending on whether the current page begins a new question or continues a question that started on an earlier page (and, if one continues onto the current page, to say what the number of that question is). Macros are also defined so that the headers and footers can vary depending on whether a question is complete on the current page or continues on to the next page (and, if one continues, to say what the number of that question is).
- You can have a horizontal rule at the base of the header and/or at the top of the footer.
- The exam can begin with one or more cover pages, which are numbered separately from the main pages of the exam and which can have headers and footers different from the ones in the main pages of the exam.
- You can include solutions in your LaTeX file and have these solutions either printed or ignored (or replaced automatically by space in which the students can write their answers) depending on a single command.

Furthermore, not stated in this excerpt, you can typeset bonus questions with bonus points and grading tables with bonus points.

You can load the `exam` document class the usual way. An example might be:

```
1 \documentclass[a4paper,12pt,addpoints]{exam}
```

which loads the `exam` document class. The paper size is set to A4 (297 mm × 210 mm, 11.7 in × 8.3 in), the document is typeset with a 12 points font and the question points are calculated.

2.1 Typesetting multiple choice questions

Then, within the document body and between `\begin{questions}` and `\end{questions}`, you enter the questions. Only multiple choice questions are considered here. The `exam` document class provides four types of multiple choice question environments:

`choices` The given choices are typeset in a linear, vertical list. Each given choice is prepended with a label name which can be set to uppercase letter, lowercase letter,

Roman numerals (uppercase and lowercase) and the Greek alphabet[2].

oneparchoices The given choices are typeset in a linear, horizontal list. Long lists are split over multiple lines. Each given choice is prepended with a label name which can be set to uppercase letter, lowercase letter, Roman numerals (uppercase and lowercase) and the Greek alphabet.

checkboxes The given choices are typeset in a linear, vertical list. Each given choice is prepended with a checkbox, which defaults to a big circle.

oneparcheckboxes The given choices are typeset in a linear, horizontal list. Long lists are split over multiple lines. Each given choice is prepended with a checkbox, which defaults to a big circle.

Within the environments, the commands `\choice` and `\CorrectChoice` designate the typesetting material. The difference between the two commands is discussed in Section 2.6.

Examples of the four environments are given below.

2.2 The `choices` environment

An example of a question with the `choices` environment is:

```
\question[5] What is the result of $1+1$?

\begin{choices}
\choice 1
\CorrectChoice 2
\choice 3
\choice 4
\end{choices}
```

which is typeset as:

1. (5 points) What is the result of $1 + 1$?

   A. 1

   B. 2

   C. 3

   D. 4

2.3 The `oneparchoices` environment

An example of a question with the `oneparchoices` environment is:

```
\question[5] What is the result of $2+2$?

\begin{oneparchoices}
\choice 1
\choice 2
```

[2]Provided by the `exam` document class.

```
\choice 3
\CorrectChoice 4
\end{oneparchoices}
```

which is typeset as:

2. (5 points) What is the result of $2+2$?

   A. 1 B. 2 C. 3 D. 4

Furthermore, the `exam` document class provides two ways of typesetting checkbox questions.

2.4 The `checkboxes` environment

An example of a vertically aligned checkbox environment is:

```
\question[5] What is the result of $1+2$?

\begin{checkboxes}
\choice 1
\choice 2
\CorrectChoice 3
\choice 4
\end{checkboxes}
```

which typesets to:

3. (5 points) What is the result of $1+2$?

   ◯ 1
   ◯ 2
   ◯ 3
   ◯ 4

2.5 The `onecheckboxes` environment

The `oneparcheckboxes` environment typesets the choices in al linear, horizontal way. An example is given below:

```
\question[5] What is the result of $1+2$?

\begin{oneparcheckboxes}
\choice 1
\choice 2
\CorrectChoice 3
\choice 4
\end{oneparcheckboxes}
```

which typesets to:

4. (5 points) What is the result of $1+2$?

◯ 1 ◯ 2 ◯ 3 ◯ 4

Other types of questions are not considered here.

2.6 Typesetting solutions

Please note the use of the `\choice` and `\CorrectChoice` commands. A `\choice` typesets as an item in the list with no special markup. A `\CorrectChoice` typesets an item in the list with special markup if the `exam` document class option `answers` is given, otherwise it typesets the same way a `\choice`. This special markup defaults to boldface for the `choices` and `oneparchoices` environments:

```
\question[5] What is the result of $2+2$?

\begin{choices}
\choice 1
\choice 2
\choice 3
\CorrectChoice 4
\end{choices}
```

which is typesets as:

5. (5 points) What is the result of $2+2$?

A. 1

B. 2

C. 3

D. 4

Note that item D is typeset in boldface. The `checkboxes` and `oneparcheckboxes` environments use a *surd*[3]:

```
\question[5] What is the result of $1+2$?

\begin{oneparcheckboxes}
\choice 1
\choice 2
\CorrectChoice 3
\choice 4
\end{oneparcheckboxes}
```

which typesets to:

6. (5 points) What is the result of $1+2$?

[3] A surd is a simplistic form of a square root sign.

◯ 1 ◯ 2 √ **3** ◯ 4

3 Using the package `exam-randomizechoices`

Although the `exam` document class is a very powerful tool to create exams, it does not provide options to typeset the content of the standard multiple choice environments in a random order[4]. This package addresses this situation.

3.1 New multiple choice environments

Basically, this package provides the user with four new multiple choice environments:

`randomizechoices` This is the randomizing counterpart of the `choices` environment. It typesets the given items in a random order.

`randomizeoneparchoices` This is the randomizing counterpart of the `oneparchoices` environment. It typesets the given items in a random order.

`randomizecheckboxes` This is the randomizing counterpart of the `checkboxes` environment. It typesets the given items in a random order.

`randomizeoneparcheckboxes` This is the randomizing counterpart of the `oneparcheckboxes` environment. It typesets the given items in a random order.

3.2 Using the new multiple choice environments

We will discuss the `randomizechoices` environment only. The other environment work alike.

You can use the new multiple choice environments in the same way as the non-randomizing counterparts. So an example of the `randomizechoices` environment might be:

```
\question[5] What is the result of $1+1$?

\begin{randomizechoices}
\choice 1
\CorrectChoice 2
\choice 3
\choice 4
\end{randomizechoices}
```

which *possibly* is typeset as:

7. (5 points) What is the result of $1 + 1$?

   A. 2

   B. 3

   C. 1

   D. 4

[4] It also doesn't provides options to randomize questions.

Here we can see that the resulting output is typeset in a different order then the choices are given. We say *possibly* because the output depends on the state of the pseudo random generator (see Section 3.7).

3.3 Arguments to the new multiple choice environments

The new multiple choice environments accept (a combination of) the following optional arguments which are local to the environment currently being typeset:

randomize The typesetting material is randomized. This is the default behaviour of the package.

norandomize Randomization is turned off. Useful if you wish to see the typesetting in the given order.

keeplast The last given item in the entered order is not part of the randomization process. This way you can keep the last item always the last item.

nokeeplast The last given item is used in the randomization process. This is the default behaviour of the package.

Sometimes you want the last given item to stick on its place. This is useful if you want to use a choice item if none of the other choices are correct:

```
\question[5] What is the result of $2+5$?

\begin{randomizechoices}[keeplast]
\choice 1
\choice 2
\choice 3
\CorrectChoice None of the above answers is correct.
\end{randomizechoices}
```

which possibly is typeset as:

8. (5 points) What is the result of $1 + 1$?

   A. 1

   B. 3

   C. 2

   D. None of the above answers are correct.

Note that the last item can also be a `\choice` command. Also note that if randomization is turned off, the `keeplast` option has no effect.

3.4 Loading the package

The package is loaded using the well-known `\usepackage` command:

```
\usepackage[ option list ]{exam-randomizechoices}
```

The package depends on the `exam` document class being loaded beforehand. If this is not the case, the package will throw an error and stops the compilation immediately.

3.5 Package options

The options in *option list* can be any combination of:

randomize This option globally turns on the randomizing of the choices given for all available typesetting environments. Randomization is turned on by default.

norandomize This option globally turns off the randomizing of the choices for all available typesetting environments. This option is useful for inspecting the resulting PDF output file with typesetting the choices in the order they were entered.

keeplast This option globally turns on the preservation of the last entered item in the new environments.

nokeeplast This option globally turns off the preservation of the last entered item in the new environments. This is the default behaviour.

overload This option makes the standard multiple choice environments behave the same as the new environment counterparts, i.e. the the standard multiple choice environments are overloaded (or redefined). This is useful if you wish to use an old exam and randomize the choices of the questions.

nooverload This option suppresses the overloading of the standard multiple choice environments so you have to use the new multiple choice environments if you want to randomize the choices to the questions. Overloading is turned off by default.

`debug` This option causes the package to emit a lot of debug messages. The messages are written to the log file by the `\PackageWarning` command. Most IDE's, such as TeXMaker, will display the messages in the transcript pane. Debug is turned off by default. There is no `nodebug` option.

If you load the package with no options, it behaves as:

```
1 \usepackage[randomize,nokeeplast,nooverload]{exam-randomizechoices}
```

3.6 Overloading the standard multiple choice environments

If the package option `overload` is in effect, the standard multiple choice environments are overloaded (or redefined) by the new multiple choice environments, i.e. the standard multiple choice environment behave the same as their randomizing counterparts. This way the user can typeset old exams or when creating an exam from a repository of (old) questions. Please note that the overloaded multiple choice environments now accept arguments just as their randomizing counterparts do.

3.7 Seeding the pseudo random generator

To get a consistent randomization, you must seed the pseudo random generator with the same seed every time you compile your document. You can set the seed using the

`\setrandomizerseed` macro. The macro has a mandatory argument that is an integer between 0 and $2^{31}-1$, TEX's largest integer. Internally, the PGF macro `\pgfmathsetseed` is called, and it is flagged that you applied a seed. If you fail to do so, the seeding value is `\time×\year` as stated by the PFG manual[5]. LATEX compilers keep track of time by an integer that holds the minutes counted since midnight. The integer is incremented every time the time passes a minute boundary. So the scenario can be that you compile your document a couple of times with no apparent differences between runs. But if the time passes a minute boundary, the next time you compile your document you'll see that the environment items have been rearranged.

3.8 Printing the key table

The package provides the typesetting of a basic key table in vertical direction. Please note that only the environments `randomizechoices`, `randomizeoneparchoices`, `choices` (if overloaded) and `oneparchoices` (if overloaded) can have valid keys in the key table. The `*checkboxes` environments can't have keys because of the typesetting regime used by the `exam` document class. The `*choices` environments use an internal counter to keep track of the choice currently being typeset. Using this counter, a label can be provided with a `\label` command. The `*checkboxes` environments don't use an internal counter so labelling them would lead to a reference to the current question (or part, or sub part or sub-sub part). Labelling the correct choices (with the `\CorrectChoice` command) is automatically handled by the package.

At the end of the exam, issue the commands:

```
\ifprintanswers
   \printkeytable
\fi
```

to print the key table. The `\ifprintansers` command is only true if the `exam` document class `ansers` option is set, thereby preventing accidental typesetting the key table. If an correct choice can't be labelled, the table entry will contain **??**, otherwise it will contain the used typesetting scheme (which is `\Alph` by default). The key table is typeset using the `tabular` environment, so it can be wrapped in a `table` environment.

The `\printkeytable` macro accepts an optional range as in:

```
\printkeytable[21-40]
```

It will typeset only the keys in this range. This is useful for generating side-by-side key tables. The start question number may be omitted, in which case the table starts at 1. The end question number may be omitted, in which case the table will be typeset up to the last question.

An example of a key table is presented below. Note the **??** in the Key column. This is the result of using the standard multiple choice environments (they are not overloaded, so no label is applied).

[5] Version 3.1.9a, page 1046.

Usage of `\printkeytable`

Question	Key
1	??
2	??
3	??
4	??
5	??
6	??
7	A
8	D

Usage of `\printkeytable[...-...]`

Question	Key
1	??
2	??
3	??

Question	Key
4	??
5	??
6	??

Question	Key
7	A
8	D

The text in the table header row can be changed using two command: the command `\keylistquestionname` sets the text above the question numbers, which defaults to "Question", the command `\keylistkeyname` sets the text above the keys, which defaults to "Key". A typical use could be:

```
\keylistquestionname{Exercise}
\keylistkeyname{Answer Key}
```

3.9 Saving the keys to a macro

It is possible to save the question numbers and the corresponding keys to a macro. Please note that only the environments `randomizechoices`, `randomizeoneparchoices`, `choices` (if overloaded) and `oneparchoices` (if overloaded) can have valid keys in the key list.

Saving a key list file is started by the command:

```
\savekeylist[command name]
```

The optional argument is the macro name, including the \. if none is supplied, it defaults to `\keylist` The key list itself is constructed as a comma-separated list. The question number and answer keys are separated with a '/'. This makes it easy to parse the list with PGF's `\foreach` command (using `\mykeylist` as command name):

```
\foreach \num/\key in \mykeylist {
    \textcolor{green!\num0!red} {question \num\ has key \key} \\
}
```

which typesets to:

question 1 has key ?
question 2 has key ?
question 3 has key ?
question 4 has key ?
question 5 has key ?
question 6 has key ?
question 7 has key A
question 8 has key D

3.10 Writing the keys to a file

The package provides the writing of the keys to the file. Please note that only the environments `randomizechoices`, `randomizeoneparchoices`, `choices` (if overloaded) and `oneparchoices` (if overloaded) can have valid keys in the key list.

Writing a key list file is started by the command:

```
\writekeylist[filename]{command name}
```

The optional parameter *filename* is the name of the file. If you don't provide a filename, `\jobname.keylist` will be used, where `\jobname` is the name of your LaTeX file you are compiling. The *command name* is used in the key list file. An example of a key list file is presented below. In essence, the file contains a `\gdef` command that assigns the key list to the supplied command name (in this case `\mykeylist`).

```
%
% Automatically generated key list file
% written by package exam-randomizechoices
% File written at July 31, 2021 (2021/07/31)
%
% Edits to this file are lost
% This file may safely be removed
%
\gdef\mykeylist{1/?,2/?,3/?,4/?,5/?,6/?,7/A,8/D}
\endinput
```

The key list itself is constructed as a comma-separated list. The question number and answer keys are separated with a '/'. This makes it easy to parse the list with PGF's `\foreach` command:

```
\input{\jobname.keylist}

\foreach \num/\key in \mykeylist {
    \textcolor{red!\num0!blue} {question \num\ has key \key} \\
}
```

Executing this code results in:

question 1 has key ?
question 2 has key ?
question 3 has key ?
question 4 has key ?
question 5 has key ?
question 6 has key ?
question 7 has key A
question 8 has key D

The key list file can safely be deleted. It is generated each time the `\writekeylist` command is executed. Edits to the file are lost.

3.11 There should be only one correct answer

As stated in the title of this section, each multiple choice question should have one, and only one correct answer. The packages issues a warning if a question has zero or more than one correct answer, but it doesn't stop compilation. If a question has no correct answer there will be a **??** in the printed key table and a ? in the key list file. If a question has two or more correct answers, the last correct answer being typeset will be printed in the key table and is written to the key list file.

Also note that using the *parts* environment to typeset multiple multiple choice questions is not supported because the automatic labelling mechanism supports only at the question level.

3.12 Verbatim environments

Verbatim environments such as `verbatim` and `lstlistings` are *not* supported in the new and overloaded standard multiple choice environments. This is due to the way the contents of these environments are collected. The contents is first read by TEX and spaces at the end of a line are removed, and the end-of-line character is replaced by a space. This will disrupt the formatted code in the verbatim environments. There is extensive discussion on StackExchange on how to tackle this problem. One could use a `verbatimbox` environment which puts the contents in a box, or one could use the `VerbatimOut` environment which writes the contents to a file for later inclusion. A portable solution would be to write contents to (separate) files and include them. In the following example using the `filecontents*` environment, four temporary files are created and included using the `\lstinputlisting` macro.

```
% Use \usepackage{filecontents} in preamble
%
\question[5]
Which of the four alternative computes the sum of the integers 1
to 10 (inclusive)?

\begin{filecontents*}{\jobname a.c}
sum = 0;
```

```
for (i=0; i<10; i++) {
    sum += i;
}
\end{filecontents*}
\begin{filecontents*}{\jobname b.c}
sum = 0;
for (i=1; i<11; i++) {
    sum += i;
}
\end{filecontents*}
\begin{filecontents*}{\jobname c.c}
sum = 0; i = 11;
while (i>0) {
    sum +- i;
    i--;
}
\end{filecontents*}
\begin{filecontents*}{\jobname d.c}
sum = 0; i = 0;
do {
    sum += i;
    i++;
} while (i<10);
\end{filecontents*}

\begin{randomizechoices}
\choice \lstinputlisting{\jobname a.c}\par    % please use \par
   otherwise label get clobbered
\CorrectChoice \lstinputlisting{\jobname b.c}\par
\choice \lstinputlisting{\jobname c.c}\par
\choice \lstinputlisting{\jobname d.c}\par
\end{randomizechoices}
```

which will typically typeset to:

9. (5 points) Which of the four alternative computes the sum of the integers 1 to 10 (inclusive)?

   A.
   ```
   sum = 0;
   for (i=1; i<11; i++) {
       sum += i;
   }
   ```
   B.
   ```
   sum = 0; i = 11;
   while (i>0) {
       sum +- i;
       i--;
   }
   ```

C.
```
sum = 0;
for (i=0; i<10; i++) {
    sum += i;
}
```
D.
```
sum = 0; i = 0;
do {
    sum += i;
    i++;
} while (i<10);
```

Please note that the `\verb` command is also not supported by the package but `\lstinline` command is.

3.13 Accessing the internal labels

Each `\question` command is accompanied with a label of the form `question@`x as provided by the `exam` document class, where x is the decimal number of the question. Within the the environments `randomizechoices`, `randomizeoneparchoices`, `choices` (if overloaded) and `oneparchoices` (if overloaded), a `\CorrectChoice` command is provided with a label (by this package) of the form `question@`x`@correctchoice` where x is again the decimal number of the question. The user can access these labels by using the `\ref` command as shown below:

```
1 The question number is~\ref{question@9} and the correct answer
2 is~\ref{question@9@correctchoice}.
```

which typesets to:

The question number is 9 and the correct answer is A.

3.14 Error messages

The package will throw you some error messages in case something is wrong. The message below is printed only at start up.

`The exam class is not loaded. Emergency stop!` — You didn't load the `exam` document class prior to loading the package. As a result of that, the package can't continue and stops compiling your document. Note that a document class can be based on the `exam` document class and will load it for you.

The next messages will only be printed if the new multiple choice environments are used *or* if the standard multiple choice environments are used when overloaded:

`You should NOT define \inaccessible. Emergency stop!` — The packages relies on the fact that `\inaccessible` is not defined. It is used by the list parser to designate `\CorrectChoice` and `\correctchoice`. Normally, a user doesn't define this command. As a result of that, the package can't continue and stops compiling your document.

`Something's wrong, perhaps a missing \choice or \CorrectChoice or ...` — You used text before an initial `\choice`, `\CorrectChoice` or `\correctchoice`.

Analogue to the famous missing `\item` messages. Compilation does continue.

`Cannot write key list file` — Writing of files is disabled so the key list file can't be generated. Compilation does continue.

4 Some internal details of the package

This section provides some internal details on the operation of the package.

4.1 Used packages

The package loads the following packages:

`environ` This package is used for defining the new environments. It provides the powerful `NewEnviron` and `RenewEnviron` commands.

`etoolbox` This package provides some useful commands, notably on the parsing of lists.

`pgffor` This package provides the powerful `\foreach` loop construct. In turn, this package loads the `pgfmath` package which supplies the `\pgfmathsetseed` and `\pgfmathrandominteger` commands.

4.2 Defining the `randomize*` environments

Defining the `randomize*` environments is done with the `NewEnviron` and `RenewEnviron` commands. As an example we discuss the `randomizechoices` environment.

We define this environment as follows (not overloaded):

```
\NewEnviron{randomizechoices}[1][]{
  %%
  %% Create a random list
  \erc@createrandomlist[#1]%
  %%
  %% Start the choices environment
  \begin{choices}%
    % Execute the list
    \erc@typesetchoices%
  \end{choices}%
}
```

Now the `NewEnviron` command has a very useful property in that when expanded it places its content in the command `\BODY`. So when the environment is used as in:

```
\begin{randomizechoices}
\choice one
\choice two
\CorrectChoice three
\choice four
\end{randomizechoices}
```

the \BODY command now contains (comments and newlines are stripped by TEX):

```
\choice one \choice two \CorrectChoice three \choice four
```

Using the command as in:

```
\BODY
```

simply expands the command. Now all the randomize* environments are defined this way, so we need a generic command that parses \BODY and provides a command that contains the randomized version of \BODY. This is handled by the command \erc@createrandomlist. What \erc@createrandomlist basically does is:

1. Parses the supplied options and sets internal flags for later use.
2. Replaces every occurrence of \CorrectChoice and \correctchoice in \BODY with \choice \inaccessible.
3. Disassembles \BODY into a set of commands by splitting on the list separator \choice. \choice is removed.
4. Randomizes the order of the set of commands.
5. Assembles the set of commands to the new command \erc@typesetchoices, thereby appending a \label to the correct choice. The "command" \inaccessible is replaced by \CorrectChoice. The other commands are prepended with \choice.

Of course this behaviour can be altered by options. For example, if the option norandomize is passed, \BODY is not randomized but labelling still takes place.

Please note: We need to replace \CorrectChoice with \choice \inaccessible because the list parser can only handle one list separator at a time. The very misunderstood command \inaccessible[6] is likely not to be entered by the user but if the user enters \inaccessible directly after a \choice command, this is converted to \CorrectChoice in step 5. Of course this is not a bug, but a feature.

After \erc@createrandomlist has done it's job, the command \erc@typesetchoices is simply expanded within a choices environment.

Now if the global overload option is in effect, some more trickery is needed. First, a copy of the choices environment is created using \let:

```
%% Save choices environment
\let\@oldchoices\choices
\let\end@oldchoices\endchoices
```

Using \let doesn't create problems, because the choices environment doesn't support options. See https://tex.stackexchange.com/questions/116670/. Next, the \choices environment is redefined:

[6] This sequence of characters sometimes appear in error messages generated by LATEX.

```
%% Renew the choices environment
\RenewEnviron{choices}[1][]{

  %% Create a random list
  \erc@createrandomlist[#1]

  %% Start the choices environment
  \begin{@oldchoices}
     % Execute the list
     \erc@typesetchoices
  \end{@oldchoices}
}
```

Note that now the `\choices` environment can handle the same options as the `randomizechoices` environment.

4.3 Note on the key table

The code of the key table is first collected using a series of `\gappto` and `\xappto` commands. Using these commands is far more easy then typesetting it directly from the package. See https://tex.stackexchange.com/questions/367979/ why. Notably the use of `\begin`, `\foreach` and `\hline` is problematic.

4.4 Verbatim environments

Verbatim environments are currently not supported. This is due to the way `\NewEviron` collects its contents. A `\NewEnviron` is not really an environment at all, but a command. One solution is to write the contents to a file and then include the file. This is being investigated. See https://tex.stackexchange.com/questions/51239/.

Also the `\verb` command is not supported, but `\lstinline` is.

5 A personal note

I've been using the `exam` document class for five years now. What I like most is the consistent typesetting of my exams. I don't use sub parts and sub-sub parts because in my opinion this not the right way to prepare an exam. I've written a department consistent class that provides the department's cover pages and some other typesetting trickery. This class is used by a small core of TEX users. Regrettably, the majority of my colleagues use Word *cough*.